The Ramtop

Published Quarterly by the Greater Cleveland Sinclair-Timex Users Group

Internet Addresses and some QL files for Your Computing Fun

by CHRIS FOWLER

If anyone's interested in the Internet, here are two sites with QL-related files. Entries beginning with "d" are further sub-directories,

Internet server: maya.dei.unipd.it Sub-directory: pub/pub/sinclair_QL

	-rwxr--r--	1 106	10	76 Dec 14 1992
README	drwxr-xr-x	2 106	100	512 Jun 15 15:36
C68	drwxr-xr-x	2 106	100	512 Oct 23 09:38
C68.version4	drwxr-xr-x	2 106	10	512 Jun 1614:58
CLUBS	drwxr-xr-x	2 106	10	512 Oct 13 1992
COMPRESS	drwxr-xr-x	3 106	100	512 Jan 11 1993
DME3	drwxr-xr-x	2 106	10	512 Jun 14 08:31
DOCS	drwxr-xr-x	3 106	100	512 Apr 20 1993
FSH_v2	drwxr-xr-x	2 106	100	512 Nov 25 1992
HARD_DISK	drwxr-xr-x	2 106	100	512 Dec 28 1992
KERMIT	drwxr-xr-x	2 106	1	512 Oct 13 1992
NEWZOO	drwxr-xr-x	2 106	100	512 Apr 20 1993
QHJ	drwxr-xr-x	2 106	10	512 Oct 13 1992
QLDB3	drwxr-xr-x	2 106	100	512 Oct 13 1992
QLSCR41	drwxr-xr-x	2 106	100	512 Oct 13 1992
QLTOOLS14	drwxr-xr-x	2 106	10	512 Oct 13 1992
QLUTIL	drwxr-xr-x	2 106	100	512 Oct 19 17:06
QUILL2ASCII	drwxr-xr-x	2 106	100	512 Mar 4 1993
ROGUE	drwxr-xr-x	2 106	100	512 Oct 19 17:01
SCHEME	drwxr-xr-x	2 106	10	512 May 17 11:52
TIFF	drwxr-xr-x	2 106	100	512 Feb 15 1993
XLISP	drwxr-xr-x	2 106	1	512 Oct 19 17:02

Welcome to University of Vaasa, Finland

garbo.uwasa.fi
MAIL SERVER INFORMATION
garbo-request PC PD archive mail server

A help file for the mail server at garbo.uwasa.fi can be requested by sending a 'send help' message, or a null message, to mailserv@garbo.uwasa.fi with a subject line of 'garbo-request'. (Don't include the quotes '').

* TO REPEAT: The Subject-line of the message MUST say * * 'garbo-request'.*

Internet server: garbo.uwasa.fi Sub-directory: ql

	-rw-rw----	1 ts	staff	4634 Oct 17 12:52 .
QLINDEX	-rw-rw-r--	1 ts	staff	9997 Oct 17 12:55
0news-ql	-rw-rw-r--	1 ts	staff	4724 Oct 17 12:53
QLINDEX	-rw-rw-r--	1 ts	staff	1404 Sep 27 05:19
QLUPLOAD.INF	drwxrwxr-x	2 ts	staff	1024 May 6 17:31
basprog	drwxrwxr-x	2 ts	staff	512 May 6 17:32
basrout	drwxrwx-wx	2 ts	staff	512 Aug 16 12:40
incoming	drwxrwxr-x	2 ts	staff	1024 Oct 17 12:53
lis	drwxrwxr-x	2 ts	staff	512 May 6 17:31 pas

Motorola 68060 Microprocessor

by Joe Circello and Floyd Goodrich
Microprocessor and Memory Technology Group Motorola Inc.

This article originally appeared in the IEEE Journal with illustrations and was downloaded from the QL Fidonet

The Motorola 68060 is the fourth generation microprocessor of the M68000 Family. User object code compatible with previous family members, it delivers 3 to 3.5 times the performance of the previous generation processor in this family, the 68040. Performance features include a superscalar integer unit, a high-performance floating point unit, dual 8 Kbyte on-chip caches, a branch cache, and on-chip memory management units. A streamlined design enables high-performance techniques to achieve a high level of parallel instruction execution. Improved performance at a low cost makes the 68060 an ideal processor for the mid to high range of desktop computing applications, and compatibility features enable it to easily upgrade performance of existing 68040-based systems. This paper describes the operation of the 68060.

Introduction and Overview

The 68060 is the fourth generation microprocessor of Motorola's M68000 Family of CISC microprocessors. It is a single-chip implementation that employs a deep pipeline, dual-issue superscalar execution, a branch cache, a high performance floating point unit (FPU), 8 Kbytes each of on-chip instruction and data caches, on-chip demand paged memory management units (MMUs). These features allow it to achieve execution rates of less than one clock per instruction sustained execution. In order to meet the performance goals of the 68060, instruction execution times needed to decrease, and parallel operations needed to increase over previous generations of M68000 microprocessors. A superscalar instruction dispatch micro-architecture is the most obvious feature of this increased parallelism on the 68060. Superscalar architectures are distinguished by their ability to dispatch two or more instructions per clock cycle from an otherwise conventional instruction stream. In addition to the superscalar features, this single chip has many other performance, upgrade and system integration features including:

*** 100% user-mode object code compatibility with 68040 * Dual-issue superscalar instruction dispatch implementation of M68000 architecture * IEEE Compatible on-chip FPU * Branch Cache to minimize pipeline refill latency * Separate 8 Kbyte on-chip instruction and data caches with simultaneous access * Bus Snooping * 68040 compatible bus protocol or new high-speed bus protocol * 32-bit nonmultiplexed address and data bus * Four-entry write buffer * Concurrent operation of Integer Unit, FPU, MMUs, caches, Bus Controller, and Pipeline * Sophisticated power management subsystem * Low-power 3.3V operation * JTAG Boundary Scan**

Design Targets

The design goals of the 68060 included providing a simple upgrade path for existing M68000 Family designs while also supplying a basis for Motorola's successful 68EC0x0 Family of embedded controllers and for the 68300 Family of modular integrated controllers.

Initial requirements for the targeted 68060 were to provide a factor of three performance enhancement over a 25 MHz 68040 with existing compiler technology. Architectural enhancements were to provide at least a 50% improvement while doubling clock frequency doubles performance. The performance estimates reflect analysis of existing object code; additional performance advantages are, of course, available when using compilers designed specifically for the 68060.

In addition to software compatibility, the 68060 preserves the investment in board-level ASICs by providing bus compatibility with the 68040. This supersocket approach facilitates upgrade of all existing and future 68040-based systems.

The 68060 uses approximately 2.4 million transistors. The part is a static CMOS design based on a 0.5 um triple level metal wafer process. This process will enable the 68060 to operate at a 3.3 volt power supply-- a greater than 50% power reduction over a 5.0 volt power supply. Since the 68060 minimizes power dissipation through a variety of architectural and circuit techniques, it is able to offer high performance processing to the laptop and portable markets in addition to the traditional computer-system markets.

Architectural Features

The architecture of the 68060 revolves around its novel integer unit pipeline. Taking advantage of many of the same performance enhancements used by RISC designs as well as developing new architectural techniques, the 68060 harnesses new levels of performance for the M68000 Family.

The superscalar micro-architecture actually consists of two distinct parts: a four-stage instruction fetch pipeline (IFP) responsible for accessing

the instruction stream and dual four-stage operand execution pipelines (OEPs) which perform the actual instruction execution. These pipeline structures operate in an independent manner with a FIFO instruction buffer providing the decoupling mechanism. A branch cache minimizes the latency effects of change of flow instructions by allowing the IFP to detect changes in the instruction prefetch stream well in advance of their actual execution by the OEPs.

The 68060 is a full internal Harvard architecture. The instruction and data caches are designed to support concurrent instruction fetch and operand read and operand write references on every clock cycle. This organization coupled with a multi-ported register file provide the necessary bandwidth to maximize the throughput of the pipelines. The operand execution pipelines operate in a lock-stepped manner that provides simultaneous, but not out-of-order, program execution. The net result is a machine architecture invisible to existing applications providing full support of the M68000 programming model including precise exceptions.

The 68060 external bus interface provides a superset of 68040 functionality. Maintaining 32-bit widths on both the address and data bus as well as a bursting protocol for cacheable memory, the 68060 supports transfers of one, two, four, or 16 bytes in a given bus cycle. The system designer can, however, choose to operate in one of two modes: a mode compatible with the 68040 protocol or a new mode consistent with higher frequency bus designs. By allowing this choice, the 68060 can easily fit into upgrades of existing designs as well as new high frequency implementations.

Pipeline Organization

The IFP is responsible for prefetching instructions and loading them into the FIFO instruction buffer. One key aspect of the design is the branch cache, which allows the IFP to detect changes in the instruction stream based on past execution history. This allows the IFP to provide a constant stream of instructions to the instruction buffer to maximize the execution rates of the OEPs. The IFP is implemented as a four-stage design.

The four stages of the IFP are: Instruction Address Generation (IAG), Instruction Cache (IC), Instruction Early Decode (IED) and Instruction Buffer (IB). The instruction and branch caches are integral components of the IFP.

Four operations can be occurring concurrently in the IFP. The IAG stage calculates the next prefetch address from a number of possible sources. The variable length of the M68000 Family instruction set as well as change-of-flow detection make this stage critical to the performance of the 68060. After the IAG sends the appropriate address to the Instruction cache, the IC stage of the IFP is responsible for performing the cache lookup and fetching the bit pattern of the instruction. The IED stage of the pipeline analyzes the bytes fetched from the instruction stream and builds an extended operation word. This lookup stage effectively converts the variable-length instruction with multiple formats into a fixed-length extended operation word that is used by the OEPs in all subsequent processing. At the conclusion of the IED stage, the prefetched bytes along with the extended operation word issue into the instruction buffer The IB stage reads instructions from the 96-byte FIFO buffer and loads them into the dual OEPs. The FIFO effectively decouples the operation of the IFP from the operations of the dual OEPs.

Consecutive instructions issue from the FIFO instruction buffer into the instruction registers of the dual OEPs. The operand execution pipelines, known as the primary OEP (pOEP) and the secondary OEP (sOEP), are partitioned into a 4-stage implementation. The four stages of the OEPs are: Decode and Select (DS), operand Address Generation (AG), Operand Cycle (OC) and the EXecute cycle (EX). For instructions writing data to memory, there are two additional pipeline stages: the Data Available (DA) and Store (ST) cycles.

The Decode and Select stage of the OEPs provides two primary functions: this stage determines the next state for the entire operand pipeline and selects the components required for operand address calculation. To determine the next state of the OEPs, the DS cycle logic tests the extended operation words to ascertain the number of instructions that can issue into the AG stage. If multiple instructions can issue into the AG stages in parallel, the first and second instructions move into the respective AG stages. If only a single instruction can issue because of architectural constraints, the first instruction issues into the pOEP, and the DS stage evaluates the second and third instructions as a pair during the next clock cycle. The net effect is a sliding 2-instruction window to examine possible pairs of instructions for parallel execution. A dedicated adder located in the AG stage sums the three components of the effective address: the base, the index and the displacement.

The Operand Cycle (OC) of the OEPs performs the actual fetch of operands required by the instruction. For memory operands, the OEP accesses the data cache in this cycle to retrieve the data. For register operands, the OEP accesses the register file containing all the general-purpose registers during the OC stage. At the conclusion of the OC cycle, the execute engines receive the required operands. The EXecute cycle (EX) performs the operations required to complete the instruction execution including updating the condition codes. If the destination of the instruction is a data or an address register, the result is available at the end of the EX stage; if the destination is a memory location, the operation requires two additional cycles. First, there is a Data Available (DA) stage where the destination operand issues to the data cache,

which aligns the operand. Second, updates to the data cache occur during the STore (ST) cycle. Additionally, there is a four longword FIFO write buffer that is selectable on a page basis and serves to decouple the operation of the OEPs from external bus cycles.

Since this is an order-two superscalar machine (dual instruction issue), the sOEP is conceptually a copy of the pOEP. A notable exception to this concept is the fact that the sOEP executes only a subset of the complete instruction set. As an example, the floating point execute engine resides only in the pOEP. Consequently, all floating point instructions must execute only the pOEP. As instructions travel down the OEPs, they remain lock-stepped. This insures that there is no out-of-order execution and thus greatly simplifies support for the precise exception model of the M68000 Family. The micro-architecture of the 68060 supports a number of optimizations to increase the number of superscalar instruction dispatches. In internal evaluations of traces from existing object code totaling several billion instructions, 50% to 60% of instructions execute as pairs.

From the preceding discussion concerning the operand pipeline stages, all data cache read references occur in the OC stage while data cache write references occur in the ST stage. The data cache uses a 4-way interleaving scheme to allow simultaneous operand read and write operations from both OEPs. The data cache directories are a single-ported design. As a result, within a superscalar pair of instructions, the 68060 only allows a single operand memory reference. The data cache also supports single-cycle references of 64-bit double-precision floating-point operands.

A common drawback to long pipelines is the penalty associated with refilling the pipeline when a change of program flow occurs. Condition code evaluation occurs in the EX stage, but waiting for a branch instruction to reach this point needlessly restricts performance. Instead, the 68060 contains a 256-entry Branch Cache (BC) which predicts the direction of a branch based on past execution history well in advance of the actual evaluation of condition codes.

The BC stores the Program Counter value of change-of-flow instructions as well as the target address of those branches. The BC also uses some history bits to track how each given branch instruction has executed in the past. The 68060 checks the BC during the IC stage of the IFP, the same stage that performs the lookup into the Instruction Cache. If the BC indicates that the instruction is a branch and that this branch should be predicted as taken, the IAG pipeline stage is updated with the target address of the branch instead of the next sequential address. This approach, along with the instruction folding techniques that the BC uses, allow the 68060 to achieve a zero-clock latency penalty for correctly predicted taken branches.

If the BC predicts a branch as not-taken, there is no discontinuity in the instruction prefetch stream. The IFP continues to fetch instructions sequentially. Eventually, the not-taken branch instruction executes as a single-clock instruction in the OEP, so correctly predicted not-taken branches require a single clock to execute. These predicted as not-taken branches allow a superscalar instruction dispatch, so in many cases, the next instruction executes simultaneously in the sOEP.

The 68060 performs the actual condition code checking to evaluate the branch conditions in the EX stage of the OEP. If a branch has been mispredicted, the 68060 discards the contents of the IFP and the OEPs, and the 68060 resumes fetching of the instruction stream at the correct location. To refill the pipeline in this manner, there is a seven-clock penalty for a mispredicted branch. If the BC correctly predicted the branch, the OEPs execute seamlessly with no pipeline stalls. Internal studies of the prediction algorithm used on the 68060 show greater than 90% accuracy from statistics gathered from several billions of instructions from applications across many runtime environments.

Floating Point Unit

The floating point unit (FPU) of the 68060 provides complete binary compatibility with previous M68000 Family floating point solutions. The 68060 performs all internal operations in 80-bit extended precision and completely supports the IEEE 754 floating point standard.

Conceptually, the FPU appears as another execute engine in the EX stage of the pOEP. A 64-bit data path between the data cache and the FPU optimizes the FPU for single-cycle references of 32- or 64-bit memory operands. As previously noted, all floating point instructions must execute through the pOEP. However, integer instructions can be simultaneously dispatched into the sOEP with most FPU instructions, and the 68060 supports overlap between the integer execute engines and the FPU. Once a multi-cycle FPU instruction is dispatched, the pOEP and sOEP continue to dispatch and complete integer instructions (including change-of-flow instructions) until another FPU instruction is encountered. At this point, the OEPs stall until the FPU execute engine is available for the next instruction.

The FPU's internal organization consists of three units: the adder, the multiplier and the divider. The 68060's design does not support concurrent floating point execution; only one of these functional units is active at a time. Table 1 shows execution times for the 68060 FPU.

Instruction	**CPU Clocks**
FMOVE	**1**

FADD	3
FMUL	4
FDIV	24
FSQRT	66

Table 1 - 68060 Floating Point Execution Times

Pipeline Example

An example of the 68060 pipeline operation is the code (not shown) which comes from a commercially available compiler and represents the inner SAXPY loop from the matrix300 program from the SPEC89 benchmark suite. Since the OEPs are decoupled from the IFP, this example only focuses on the OEPs. This loop executes 13 instructions in only ten clock cycles, producing a steady-state performance of 0.77 clocks per instruction (CPI). This code includes two multi-cycle FPU instructions (4-cycle FMUL and 3-cycle FADD), but the superscalar micro-architecture is able to effectively exploit the parallelism within the loop to achieve a less than one CPI measure.

This example code loop demonstrates several major architectural features of the 68060. Of the 13 instructions, the 68060 dispatches four groups of 2-instruction pairs (at cycles 1, 2, 4, 5), one group of three instructions (at cycle 9) and two individual instructions (at cycles 3 and 8). At cycle 3, the pair of instructions being examined is {pOEP = lsl.l, sOEP = fadd.d}. Since all floating-point instructions must issue into the pOEP, the fadd.d does not issue into the sOEP. On the next cycle, a new 2-instruction pair is examined {pOEP = fadd.d, sOEP = add.l}, and at this time, both instructions issue down the OEPs. At cycles 6 and 7, the pipeline stalls on the fadd.d instruction as the 4-cycle fmul completes execution. The floating-point store operation at cycle 8 inhibits any sOEP dispatch because of certain post-exception fault possibilities. At cycle 9, an instruction triplet is dispatched {add.l, subq.l, bcc.b}. Recall the branch cache utilizes various instruction folding techniques that effectively allow this predicted as taken branch to execute in 0 cycles. Finally, at cycle 10, the pipeline stalls for one clock on the floating-point store instruction as it waits for the completion of the three-cycle fadd.

Power Management On Chip

With 2.4 million transistors operating at frequencies of 50 MHz and higher, power management becomes a crucial issue on the 68060. From the inception, the 68060 focused on minimizing chip-level power dissipation. There are primarily three different areas of interest for power dissipation.

The 68060 operates from a 3.3 volt power supply. Since power dissipation is a function of the square of the power supply voltage, simply changing the power supply voltage to 3.3 volts results in a 56% reduction in power compared to a 5 volt power supply. In addition to a lower supply voltage, the 68060 is a completely static design. The 68060's operating frequency, which linearly affects chip-level power dissipation, can vary dynamically down toward the DC range. Although the 68060 is a 3.3 volt part, its I/O buffers interface to either 3 volt or 5 volt peripherals and memory, facilitating upgrades of existing designs.

Sophisticated power management circuitry on chip dynamically controls and minimizes power consumption. This circuitry selectively updates modules on the 68060 on a clock-by-clock basis, dynamically shuttung off the circuits not required to support the activities in the current clock cycle. Entire areas of the 68060 can shut off for long periods of time when they are not required.

The 68060 also incorporates the LPSTOP instruction. This instruction effectively puts the 68060 into a low-power sleep mode in which it stays until awakened by an externally generated interrupt. Data on previous members of the M68000 Family shows that use of the LPSTOP instruction can extend battery life in portable applications by over 250%.

Summary

The 68060 relies on new as well as standard architectural techniques to extend the performance of the M68000 Family product line. Performance simulations predict that between 3 and 3.5 times a 25 MHz 68040 are possible using existing object code.

The 68060 relies on a deep internal pipeline and a superscalar internal architecture coupled with 8 Kbyte instruction and data caches, a 256-entry branch cache, on-chip MMUs and an on-chip FPU to bring new levels of performance to the M68000 Family architecture.

Power management is very important on the 68060, and this design uses dynamic power management techniques to minimize power consumption. The 68060 operates from a 3.3 volt power supply, which greatly reduces its power dissipation. Although the 68060 operates at a lower operating voltage, it interfaces to both 3 volt and 5 volt peripherals and logic.

In addition to providing full application object code compatibility with previous CPUs in this family, the 68060 provides a superset of 68040 hardware functionality. Designs compatible with existing and future 68040 systems are simple, and higher frequency designs are possible using a new bus interface protocol.

Acknowledgements

The authors would like to thank all members of the 68060 design team and management Without their concerted team effort, this project and this paper would not have been possible.

References

Bernal, R.D. and Circello, J.C., "Putting RISC Efficiency To Work in CISC Architectures," VLSI Systems Design, September 1987, pp. 46-51.

Circello, J.C. et al, "Refined Method Brings Precision to Performance Analysis," Computer Design, March 1, 1989, pp. 77-82.

Edenfield, R.W. et al, "The 68040 Processor, Part 1, Design and Implementation," IEEE MICRO, February 1990, pp. 66-78.

Diefendorff, K. and Allen, M., "Organization of the Motorola 88110 Superscalar RISC Microprocessor," IEEE MICRO, April 1992, pp. 40-63.

Hennessy, J. and Patterson, D., "Computer Architecture: A Quantitative Approach," Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1990.

The Chirstmas Party and Auction was attended by the members above

The Ramtop

Spring 1994

Is published quarterly by the Greater Cleveland Timex Sinclair Users Group

Thomas Simon, Editor
Jon Kaczor, Production & Distribution

Please send editorial contributions and correspondence to 615 School Ave., Cuyahoga Falls, Ohio 44221

Sinclair Notes

If you have something that you feel we should publish write to the editor at the Ramtop. We need contributions from our readers.

Spectrum Resources

Keith Turner informs us of one firm which still sells new Spectrums and Sinclair stuff. That is **W.N.Richardson & Co (EEC), 18-21 Misbourne House, Chiltern Hill, Chalfont St Peter, Bucks, SL9 9UE, England.** If you are visiting the UK or have a friend there you can probably get one second hand for less than fifty pounds. Keith found the best place to find one is a computer small ads paper such as Micro Mart, which is published on Thursdays by **MicroMart (UK) Ltd, 24 Richmond Road, Olton, Solihull, West Midlands, B92 7RP, England.** The Spectrums in a recent Micro Mart included a 48, a 128+, and a +2 with monitor and 3.5 diskette drive, all for less than fifty pounds each. It would be worth scouring this magazine for Spectrum add-ons while they are still available too.

Spectrum History Remembered

The Spectrum range started with the cute little 16K and 48K slabs with the rubber keys. The next release was the 48+, which was no more than a 48 with the original ROM (it isn't certain that the PCB was different from the later 48s) and precisely 3 noticeable changes - a QL style keyboard, a reset button (on the side of the case, with flying leads messily soldered across the reset cap) and a much more insulting user manual. (Annoyingly, the joystick port doesn't properly connect due to the squarer, thicker case shape.)

Then came the 128. This was a real advance, with a GI sound chip, bank switched RAM and an extended basic, not to mention some interfacing capabilities (rather rudimentary serial and keypad ports, starting Sinclair's trend for BT jacks which was continued with the QL). It was still compatible with 48 software and it was quite a nice 8-bit machine, and as far as the Spectrum was ever taken.

After the onset of Amstrad, we had the originally titled Spectrum +2, effectively a 128 in a different case, similar to the Amstrad CPC range, with a cassette deck to the right of the keyboard operated by keyboard- style keys. The keyboard itself was still of

Continued Page 8

Forth for the QL

by Dirk Kutscher % MAUS HB2

I have ported the Unix version of TILE-forth by Mikael Patel to QDOS. The files (two zoo archives, 300k together) can be downloaded from sabrina.dei.unipd.it. Maus users will find it on the next 'RING' and on HB2.MAUS bbs.

Here are some extracts of one of the documentation files. (Don't expect a concise 4th-system - it's rather what I would call a monster.) But you can do interesting things with it. Because the threading systems extends the standard forth model and adds more features than the usual 'immediate' flag such as 'private' or 'compilation' it enables you to treat vocabularies as abstract data types. Most interesting are the belonging lib-files which contain various extensions such as multi-tasking with condition queues, semaphores, channels and rendezvous...

THREADED INTERPRETIVE LANGUAGE ENVIRONMENT (TILE) FORTH RELEASE 2.1, August 20, 1990 Mikael R.K. Patel Computer Aided Design Laboratory (CADLAB) Department of Computer and Information Science Linkoping University S-581 83 LINKOPING SWEDEN Email: mip@ida.liu.se

1. INTRODUCTION

TILE Forth is a 32-bit implementation of the Forth-83 Standard written in C. Thus allowing it to be easily moved between different computers compared to traditional Forth implementations in assembly.
Most Forth implementations are done in assembly to be able to utilize the underlying architecture as optimal as possible. TILE Forth goes another direction. The main idea behind TILE Forth is to achieve a portable forth implementation for workstations and medium size computer systems so that new groups of programmers may be exposed to the flavor of an extensible language such as Forth.

The implementation of TILE Forth is selected so that, in principle, any C-level procedure may become available on the interactive and incremental forth level. Other models of implementation of a threaded interpreter in C are possible but these are not as flexible.

TILE Forth is organized as a set of modules to allow the kernel to be used as a general threading engine for C. Environment dependencies such as memory allocation, error handling and input/output have been separated out of the kernel to increase flexibility. The forth application is "just" an example of how to use the kernel.

Comparing forth implementations using the traditional benchmarks such as the classical sieves calculation is difficult because of the difference in speed between workstations and personal computers. The Byte sieves benchmark is reported to typically run in 16 seconds on a direct threaded forth implementation. This benchmark will run in 17 seconds in TILE forth (compiled with GNU CC and optimized) on a SUN-3/60 and less than 9 seconds on a SUN SPARCstation 1. These times are the total time for loading TILE forth, compiling and executing the benchmark. Comparing to, for instance, other interpretive languages such as Lisp, where one of the classical benchmarks is calculation of the Fibonacci function, the performance increase is over a magnitude.

The kernel supports the Standard Forth-83 word set except for the blocks file word set which are not used. The kernel is extended with many of the concepts from modern programming languages. Here is a list of some of the extensions; argument binding and local variables, queue management, low level compiler words, string functions, floating point numbers, exceptions and multi-tasking. The TILE Forth environment also contains a set of reusable source files for high level multi-tasking, data description and structuring modules, and a number of programming tools.......

2. EXTENSIONS

What is new in TILE forth? First of all the overall organization of words. To increase portability and understanding of forth code modules vocabularies are used as the primary packaging mechanism. New data types such as rational and floating point numbers are implemented in separate vocabularies. The vocabularies act as both a program module and an abstract data type.

2.1 Extensible interpreter

To allow extension of the literal symbol set (normally only integer numbers) each vocabulary is allowed to have a literal recognition function. This function is executed by the interpreter when the symbol search has failed. The literal recognizer for the forth vocabulary is "?number". This simple mechanism allows modules such as for rational and floating point numbers, and integer ranges to extend with their own literal function.

2.2 Data description

As the Forth-83 Standard lack tools for description of data structures TILE Forth contains a fairly large library of tools for this purpose. These are described more in detail in the next section.

2.3 Argument binding and local variables

When writing a forth function with many arguments stack shuffling becomes a real pain. Argument binding and local variables is a nice way out of these situations. Also for the new-comer to Forth this gives some support to this at first very cryptic language. Even the stack function may be rewritten using this mechanism:

```
: 2drop { a b } ;
: 2swap { a b c d } c d a b ;
```

: fac { n } n 0> if n 1- recurse n * else 1 then ;

The argument frame is created on top of the parameter stack and is disposed when functions is exited. This implementation style of reduces the cost of binding as most functions have more arguments then return values. A minimum number of data elements have to be move to create and manage the argument frame.

2.4 Exception handling

Another extension in TILE Forth is exception handling with multiple exception handling code block. The syntactical structure is very close to that of Ada, i.e., any colon definition may contain an error handling section. Should an error occur during the execution of the function the stack status is restore to the situation at the call of the function and the latest exception block is executed with the signal or exception as a parameter;

```
exception zero-divide ( -- exception)
: div ( x y -- z)          / exception> ( x y signal -- )
drop zero-divide raise     ;
```

Error situations may be indicated using an exception raise function. Low level errors, such as zero division, are transformed to exceptions in TILE Forth.

2.5 Entry visibility and forward declaration

Last, some of the less significant extension are forward declaration of entries, hidden or private entries, and extra entry modes. Forward declaration of entries are automatically bound when the entry is later given a definition. Should a binding not exist at run-time an error message is given and the computation is aborted.

```
forward eval ( ... )
: apply ( ... ) ... eval ... ; : eval ( ... ) ... apply ... ;
```

Three new entry modes have been added to the classical forth model (immediate). These allow hiding of entries in different situations. The first two marks the last defined words visibility according to an interpreter state. These two modifiers are called "compilation" and "execution" and are used as "immediate". A word like "if" is "compilation immediate" meaning it is visible when compiling and then always executed.

```
compiler forth definitions
: if ( -- ) compile (?branch) >mark ; compilation immediate
```

The "private" modifier is somewhat different. It concerns the visibility across vocabularies (modules and types). If a word is marked as "private" the word is only visible when the vocabulary in which it is defined in is "current". This is very close to the concept of hidden in modules and packages in Modula-2 and Ada.

```
4 field +name ( entry -- addr) private
```

The above definition will only be visible in the vocabulary it was defined. The "private" modifier is useful to help isolate implementation dependencies and reduce the name space which also increases compilation speed. ...

3. SOURCE LIBRARY

The TILE Forth programming environment contains a number of tools to make programming in Forth a bit easier. If you have GNU Emacs, TILE Forth may run in a specialized forth-mode. This mode supports automatic program indentation (pretty printing), documentation search, and interactive and incremental program development, or "edit-compile-test" style of program development.

To aid program development there is also a source code library with manual pages, documentation (glossary), and test and example code. Most of the source code are data modeling tools. In principle, from bit field definition to object oriented structures are available. The source code library also contains debugging tools for tracing, break- point'ing and profiling of programs.

The first level of data modeling tools are modules for describing; 1) bit fields, 2) structures (records), 3) aggregates of data (vectors, stacks, buffers, etc), 4) high level data objects (lists, sets, etc), and last, 5) object oriented programming with the three major models (relations, prototypes, and classes/instances).

The next level of tools are some tools for high level syntactic sugar for multi-tasking concepts (semaphores, channels, etc), anonymous code block (blocks), a general top down parser with backtrack and semantic binding, and a simulation package. The source library will be extended during the coming releases.

4. PROGRAMMING STYLE

5. SOURCE FILES

The TILE Forth source is broken down into the following files:

README This short documentation of TILE.
COPYING The GNU General Public License.
INSTALL Some help on how to install TILE Forth.
PORTING Some help on how to port TILE Forth and typical problems.
Makefile Allows a number of compilation styles for debugging, profiling, sharing etc. New machines and conditional compilation symbols are added here.
src The C source library with the kernel code and
GNU Emacs forth-mode E-lisp source.??
lib The Forth-83 source library for data description and management, high level tasking, etc.
tst Test and example file for each Forth-83 source code file and a set of benchmarks.
man Manual pages for the TILE Forth C kernel and Forth-83 source code library.
doc Documentation and glossaries for each source code file and kernel vocabularies (generated by make help command).
bin Utility commands and the TILE forth

compiler/interpreter.

6. CONFIGURATION 7. COPYING

This software is offered as shareware. You may use it freely, but if you do use it and find it useful, you are encouraged to send the author a contribution (>= $50) to the following address:

TILE Technology HB Stragatan 19 S-582 67 Linkoping SWEDEN

If you send me a contribution, I will send you the manual pages and documentation files (and paper copies if you don't have access to a good laserprinter), and will answer questions by mail. Your name will also be put on a distribution list for future releases.

Notes Continued

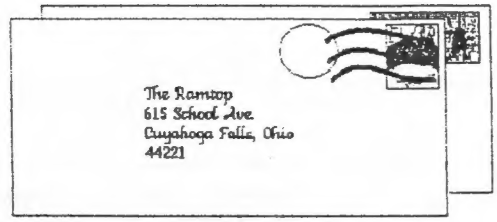
membrane construction, but had a more typewriter like quality thanks to it's raised keys. There were 2 releases, the +2 and the +2a, which had a minor bug fix, something to do with the tape deck. Both of these machines had the nice interfaces on the 128 removed and replaced with a couple of Amstrad joystick ports, which, needless to say, were incompatible with everyone else's. The +2a was a +3 without a disc drive. It had the black case instead of the +2 grey one. The main interface port was different, which meant some incompatibility with peripherals and software.

The final "official" Spectrum (rather than semi-clones built both in the UK and the USSR) was the +3, which continued the descent into Amstraddom by adding the 3" drive from the CPC6128 where the tape drive in the +2 went. The 3" format had the dual disadvantages of the lack of use by other manufacturers (and where it was used, it was used in a different and usually better way), and the lack of an existing software base in that format, and so the system was generally found to be a lot less useable than existing OEM external drives, which usually provide "snapshot" capabilities.

Publish and Perish

Pete Collier reports that both Your Sinclair & Sinclair User are dead. Both died early on in 1993. Sinclair User had teamed up with Crash, but they still couldn't survive. Your Sinclair did much better, but once the big names (Ocean, US Gold, Gremlin, Virgin) pulled out of the 8 bit market, they had no new games to review and perished within months.

A number of Sinclair newsletter/fanzines have started up and one of these is called ZAT which is being published bi-monthly. If anybody wants more info on it, send an SAE with 2 international reply coupons (available at your local post office) to **ZAT,33 Dawley Bank, Dawley, Telford, Shropshire, TF4 2LQ.**



What ever happened to Uncle Clive ?

Not having heard from Uncle Clive lately and not having seen him with Don King, some may have thought his picture would soon be seen at breakfast on a milk carton. Good news! Apparently Sir Clive and his son are running a mail order company for PC stuff. According to Scott Teleford of the the University of Edinburgh, Clive and his son, Crispin, have set up a company called Sinclair Games, which sells discount PC games, software, joysticks, and the usual stuff, by mail order. Clive is a company director. There's a rather cheesey full page ad in last month's PC Review magazine (the one with the Frontier review and the pink cover), and probably in other PC games mags too.

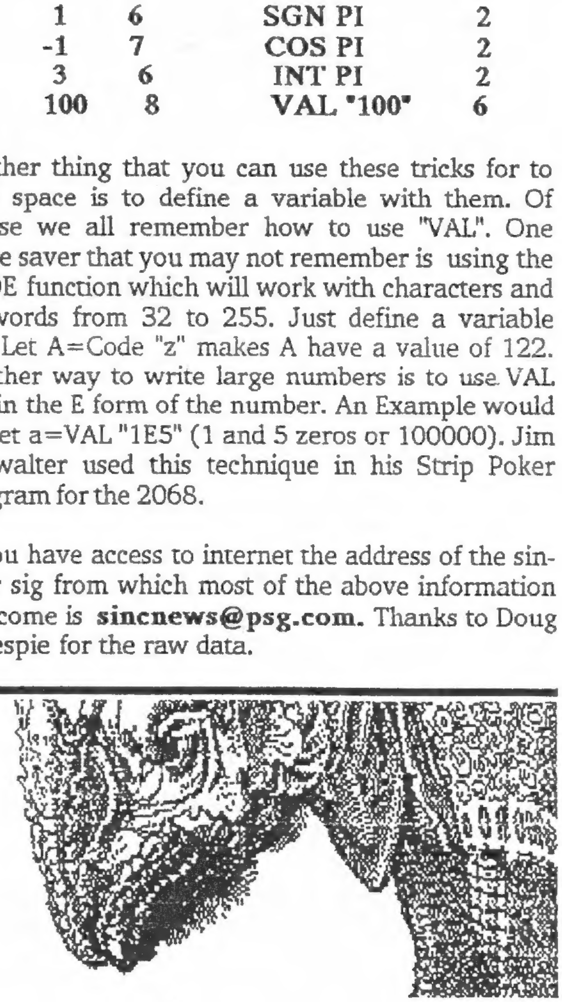
Another Blast from the Past

For 1K programmers, do you remember these memory saving tricks:

number	bytes	solution	bytes
0	6	SIN PI	2
1	6	SGN PI	2
-1	7	COS PI	2
3	6	INT PI	2
100	8	VAL "100"	6

Another thing that you can use these tricks for to save space is to define a variable with them. Of course we all remember how to use "VAL". One space saver that you may not remember is using the CODE function which will work with characters and keywords from 32 to 255. Just define a variable like Let A=Code "z" makes A have a value of 122. Another way to write large numbers is to use VAL but in the E form of the number. An Example would be Let a=VAL "1E5" (1 and 5 zeros or 100000). Jim Showalter used this technique in his Strip Poker program for the 2068.

If you have access to internet the address of the sinclair sig from which most of the above information has come is **sincnews@psg.com.** Thanks to Doug Gillespie for the raw data.



President's Message

Hello to you all! I certainly hope you have not had too much trouble with this unbelievable cold weather! (Back in January when this article was written...ed) I have been fairly lucky so far. Having an outside job in this cold is no fun! The Christmas meeting was a big success! There were a lot of faces we have not seen in quite a while. Some of these include: Dale Sincovick, Jim Lewis, Ron Lutz, Dave Hoshor, Andy Korsorik, and others. Lots of good stuff was for sale and I bought some of it. Thanks to all of you for coming! I hope you won't be strangers and see us more often!

Let's move on to the subject of our meetings. We have become so informal that we really don't have any structure at all. This is mostly my fault. I would like to improve on this in the future. We should always have a treasury report and give each member a chance to bring in some new information or something of interest. Also, we really need to have some schedule for demos. Demos can be on almost anything as long as it is computer related in some way. I also have been thinking that since almost all of us now have IBM clones that maybe we should change the name of our club to reflect this. I would like for each of you to give your opinion of this. If you like the idea, think of a name and let's discuss it in a month or two. That's about it for now. Take care till next time!

A User's Guide To Computer Terminology

By Suzette Cohen

Beginner: A person who believes more than one-sixteenth of a computer salseperson's spiel.

Advanced User: A person who has managed to remove a computer from its packing materials.

Power User: A person who has mastered the brightness and contrast controls on any computer's moniter.

Sales Associate: A former cheese-monger who has recently traded mascarpone for MS-DOS

Sales Manager: Last week's new sales associate.

Consultant: A former sales associate who has mastered at least one tenth of the D-BASE 3 Plus Manual.

Systems Integrator: A former consultant who understands the term "AUTOEXEC.BAT".

Service: Cursory examination, followed by the utterance of the phrase "It can't be ours" and either of the words "hardware" or "software."

Support: The mailing of advertising literature to customers who have returned a registration card.

Alpha Test Version: Too buggy to be released to the paying public.

Beta Test Version: Still too buggy to be released.

Release Version: Alternate pronounciation of "beta test version."

Loading Spectrum Programs to your PC with the Sound Blaster

By Tomi Edgehill

The following information can be found from book "An Expert Guide to the Spectrum" written by Mike James and published by Granada. Text is also result of my experiments.

Spectrum casette file consist of two parts: header part and the actual file part. Both parts have same basic structure: first there is loader tone, then a sync pulse and after it the actual data.

Loader tone signal signal is 614.9 microsecond low and 614.9 microseconds high. The sunc pulse is 190.6 microseconds low and 210 microseconds high. In data part one bit is 488,6 microseconds low and 488.6 microsecons high. Zero bit is 244.3 microseconds low and 244.3 microsecond.

The data is stored so that most significant bit of byte is saved first and least signicant bit last. The first byte of the data part of file tells the type of file and the actual data follows after it. The last byte in data part is cheksum which is obtained by xoring all data bytes together.

And here is a simple Turbo Pascal program which I have used to load files from spectrum tapes. It works nicely when Spectrum is directly connected to SoundBlaster and I save file in spectrum and start that loading program in PC. With casette deck test have been less succesful. The program loads only the header part, but can be easily extended.

Enhanced: Less awful in some ways than the previous model, and less likely to work as expected.

Upgraded: Didn't work the first time.

Upgraded and Improved: Didn't work the second time.

Memory-Resident: Ready at the press of a key to disable any currently running program.

Multitasking: A clever method of simultaneously slowing down the multitude of computer programs that insist on running too fast.

Encryption: A powerful algorithmic encoding technique employed in the creation of computer manuals.

Desktop Publishing: A system of software and hardware enabling users to create documents with a cornucopia of typefaces and graphics and the intellectual content of a Formica slab; often used in conjunction with encryption.

High Resolution: Having nothing to do with graphics on an IBM-compatible microcomputers.

FCC-Certified: Guaranteed not to interfere with radio or television reception until you add the cable required to make it work.

American: Italian or Taiwanese, as in "American Telephone and Telegraph."

American-Made: Assembled in America from parts made abroad.

Windows: A slow-moving relation of the rodent family rarely seen near computers but commonly found in specially marked packages of display cards, turbo cards, and Grape-Nuts Cereal.

TopView: The official position of IBM brass that an abysmally slow character-based multitasking program is the product of the future.

DOS-SHELL: An educational tool forcing computer users to learn new methods of doing what they already can.

UNIX: Sterile experts who attempt to palm off bloated, utterly arcane, and confusing operating systems on rational human beings.

EMS: Emergency Medical Service; often summoned incases of apoplexy induced by attempts to understand extended, expanded, or enhanced memory specifications.

Videotex: A moribund electronic service offering people the privlege of paying to read the weather on their TV screens instead of having Willard Scott read it to them free while they brush their teeth.

Artificial Intelligence: The amazing, human-like ability of a computer program to understand that the letter y means "yes" and the letter n means "no."

Electronic Mail: A communications system with built-in delays and errors designed to emulate those of the United States Postal Service.

Turbo Card: A device that increases an older-model computer's speed almost enough to compensate for the time wasted in getting it to work.

Laser Printer: A xerographic copying machine with additional malfunctioning parts.

Workstation: A computer or terminal slavishly linked to a mainframe that does not offer game programs.

RISC: The gamble that a computer directly compatible with nothing else on the planet may actually have decent software written for it someday.

AUTOEXEC.BAT: A sturdy aluminum or wooden shaft used to coax AT hard disks into performing properly.

Plotter: A terroristic hypodermic device used to inject graphic representations of boring data into boring meetings.

Clone: One of the many advanced-technology computers IBM is begining to wish it had built.

CD-ROM: An optical device with storage sufficieent to hold billions of predictions claiming it will revolutionize the information industry.

IBM Product Centers: Historical landmarks forever memoralizing the concept of "list price only."

IBM: Somewhat like an IBM product; in current parlance, invariably followed by the word "compatible."

Fully IBM Compatible: Somewhat IBM compatible, but won't run IBM BASIC programs.

100% IBM Compatible: Compatible with most available hardware and software, but not with the blockbusters IBM always introduces the day after tomorrow.

Hard Disk: A device that allows users to delete vast quantities of data with simple mnemonic commands.

Mouse: A peripheral originally christened "vermiform appendix"because of its functional resemblance, renamed for its appropriateness as a cat toy.

Printer: An electromechanical paper-shredding device.

```
Program Spectrum_to_PC;

{ Sinclair Spectrum casette loader for IBM PC compatibles
  with Sound Blaster. Sound Blaster must be at default I/O address 220h.


  Copyright 1992 Tomi Engdahl


  Sinclair Spectrum is a registered trade mark of Sinclair Reserch Ltd.
  Sound Blaster is a register trade mark of Creative Labs Inc.

}

Uses crt;

Var
   timer_count:word;
   tmp:byte;
   old_bit:byte;

Const
   Timer0=$40;
   TimerCtlr=$43;
   TimerClk=1193180;

   Audio0=128;
   hysteresis=20;

   mid_value=2000;  {2000}

   ltone_HI=4000;


Procedure CLI;
Begin
   Inline($FA);                 {cli, disable interrupts}
End;

Procedure STI;
Begin
   Inline($FB);                 {cli, enable interrupts}
End;

Function Timer0Read:word;
Var
   value:word;
Begin
   Port[TimerCtlr]:=0;        {latch timer 0}
   value:=Port[Timer0];
   value:=value+(Port[Timer0] shl 8);
   Timer0Read:=value;
End;

Function time_difference:word;
Var value:word;
Begin
{R-}
   Port[TimerCtlr]:=0;        {latch timer 0}
 value:=Port[Timer0];
   value:=value+(Port[Timer0] shl 8);
   time_difference:=timer_count-value;
   timer_count:=value;
{R+}
End;

Function Read_SBADC:byte;
Begin
      Inline(
      $BA/$2C/$02/      {MOV     DX,022C    }
      $EC/              {IN      AL,DX      }
      $A8/$80/          {TEST    AL,80      }
      $75/$FB/          {JNZ     0107       }
      $B0/$20/          {MOV     AL,20      }
      $EE/              {OUT     DX,AL      }
      $BA/$2E/$02/      {MOV     DX,022E    }
      $EC/              {IN    AL,DX      }
      $A8/$80/          {TEST    AL,80      }
      $74/$FB/          {JZ      0112       }
      $BA/$2A/$02/      {MOV     DX,022A    }
      $EC/              {IN      AL,DX      }
      $A2/tmp);       {MOV     a,AL       }

      Read_SBADC:=tmp;
End;

{Function schmitt_trigger(value:byte):byte;
Begin
   If old_bit=0 Then If value>(audio0+hysteresis) Then old_bit:=1;
   If old_bit=1 Then If value<(audio0-hysteresihen old_bit:=0;
   schmitt_trigger:=old_bit;
End;}

Function schmitt_trigger(value:byte):byte;
Begin
   If old_bit=0 Then If value>(audio0+hysteresis) Then old_bit:=1;
   If old_bit=1 Then If value<(audio0-hysteresis) Then old_bit:=0;
   schmitt_trigger:=old_bit;
End;
```

```
Function bit_from_tape:byte;
Var
   adcvalue:byte;
   count:integer;
Begin
   Repeat Until schmitt_trigger(Read_SBADC)=0;
   Rei_difference>mid_value Then bit_m_tape:=1
   Else bit_from_tape:=0;
End;

Procedure Wait_loader_tone;
Var
   diff:word;
   count:integer;
Begin
   diff:=0;
   Repeat
       Repeat Until schmitt_trigger(Read_SBADC)=0;
       Repeat Until schmitt_trigger(Read_SBADC)=1;
       diff:=time_difference;
       If (diff>mid_value) and (diff<Ltone_hi) Then Inc(count)
       Else count:=0;
   Until count>=10;
End;


{******* High level packet routines ************* }

Var tavut:array[0..50000] of byte;

Procedure load_bytes(maara:word);
Var
   tavu,p,a,n:byte;
   laskuri:word;
Begin
   ClrScr;
   Repeat Until bit_from_tape=1;          {wait for loader tone}
   Writeln('loader tone detected');
   Repeat Until bit_from_tape=0;          {synclse wait}
   {Writeln('sync');}
   For n:=0 to 7 Do p:=bit_from_tape;     {over the start byte}
   laskuri:=0;
   Repeat
      tavu:=0;
      For n:=0 to 7 Do tavu:=(tavu shl 1)+bit_from_tape;
      tavut[laskuri]:=tavu;
      Writeln(tavu);
      Inc(laskuri);
    Until laskuri>=maara;
End;

Procedure print_header;
Var a:integer;
Begin
   Writeln;
   Write('Program: ');
   For a:=1 to 10 Do
   Begin
      Writhr(tavut[a]));
   End;
   Writeln;
   Writeln('Start:   ',tavut[13]+256*tavut[14]);
   Writeln('Length:  ',tavut[11]+256*tavut[12]);
End;

Procedure header_load;
Begin
   load_bytes(17);
   print_header;
End;


{ ************* Main program ************** }

 dummy:word;
Begin
   CLI;
   old_bit:=0;
   dummy:=time_difference;
   Wait_loader_tone;
   header_load;
   STI;
   Repeat Until KeyPressed;
End.
```

The ZX Spectrum 48/128 Emulator For IBM & Compatables: Z80 Version 2.01

Turn your PC into a real ZX Spectrum 48/128!
The fastest, most compatible and most complete emulator available! Main features:

- Full Spectrum emulation, border, flash, beeper, Interface I, Microdrive in cartridge file, RS232 input and output redirection to file, COM or LPT, joystick support, 128K sound through Soundblaster or internal speaker, built-in monitor,
- Able to load ANY, even protected or speed-saved program from tape, to save to tape, to redirect tape loads and saves to disk for easy file access,
- 2500 line English documentation, frequently-asked-questions file, PostScript file of doc, keyboard help screen, utilities to convert Spectrum screens to .GIF and .PCX files, convert snapshot files and tape files from 5 other Spectrum emulators to own format and vv., to read DISCiPLE and +D disks,
- Z80 processor emulation including R register, inofficial instructions, inofficial flags,
- Runs okay under DOS, Windows and DesqView,
- Full source code of emulator and utilities included!

Runs on any 640K PC; too slow for practical use on PC/XT's but fast enough on AT's; runs at about 100% on 16MHz AT's (can be slowed down on faster machines), uses VGA/EGA/CGA or Hercules.

This program costs US$ 20. You will receive a 3.5" DD disk (5.25" disks on request), and you'll be kept informed about updates. Please send bank notes, name and address to:

Gerton Lunter
P.O. Box 2535
NL-9704 CM Groningen
The Netherlands

If you send a cheque, please add US$ 15. Please allow 4 weeks for delivery.

The Ramtop
4568 Williamston Ave.
Brooklyn, Ohio 44144